Universal Resource Interface Module (URIM) for the Joint Force Protection Advanced Security System (JFPASS)

S.H. Cutler, J.R. Cruickshanks, C.M. Barngrover, T.A. Kramer, A.F. Nans
Space and Naval Warfare Systems Center, Pacific (SSC Pacific)
53560 Hull Street, San Diego, CA 92152

ABSTRACT

The Joint Force Protection Advanced Security System (JPFASS) is a Department of Defense effort to improve conventional force protection. It is sponsored and managed by Joint Program Manager - Guardian (JPM-G). The main objective of JFPASS is to provide an integrated and layered base defense system, which includes data fusion, Command and Control (C2) nodes, Common Operation Picture (COP) nodes, and full integration of a selected range of robots, sensors, cameras, weapons, tracking systems, and other C2 systems. The URIM is the main integration tool for several sensors, cameras, and weapons in JFPASS.

The Universal Resource Interface Module (URIM) is an extremely flexible framework for rapidly integrating new sensors into the JFPASS. Each sensor system has its own proprietary protocol, which makes integration high cost and risk. The URIM communicates directly with each sensor system though a protocol module and maintains a generic data object representation for each sensor. The URIM then performs a translation of the data into a single protocol, in this case Systems Engineering and Integration Working Group (SEIWG) ICD-0100. With this common protocol the data can be provided to a data server for publishing. Also, this allows for network control and management of all sensor systems via any C2 node connected to the data server.

Keywords: FIRRE, JBC2S, MOCU, force protection, command and control, unattended ground sensor, unmanned ground vehicle, robotic architecture,

1. BACKGROUND

The purpose of JFPASS is to provide comprehensive, effective, and sustainable joint force protection capabilities. There are many systems in existence that fill force protection needs in reliable and proven ways. JFPASS is a combination of all these systems, brought together through integration. There is a central data fusion engine which receives all data from the various systems and publishes it to all subscribers (including COP nodes and C2 nodes) in a "fused" manner, meaning data passed on to the C2 software is correlated while redundant and extraneous information is left out. For example, if three separate radars are tracking the same vehicle, the data fusion engine only sends one track to the COP. The data fusion engine can also trigger automated responses based on received data. One example would be the automatic slewing of the nearest available camera to assess a new radar track. These capabilities can significantly increase the security of a base while greatly reducing the manpower and costs of force protection. However, reliable data must first be obtained from each of the many integrated systems, as well as reliable control capabilities for automatic responses.

The JFPASS system of systems leverages many of the lessons learned from the Force Protection Joint Experiment (FPJE) held at Eglin Air Force Base from March 2007 thru March 2008. During the FPJE, it became quickly apparent that integrating new systems into the central data fusion engine could be very risky and very time consuming. The integrators realized that much of the work done was duplication, as each commercial or government product required individual translation between its native protocol and the standard SEIWG ICD-0100 interface, used by the data fusion engine and all C2 nodes and COP nodes in the FPJE system. Each piece of software that performed this translation was known during the Experiment as a RIM (Resource Interface Module).

One brilliant programmer saw a way to reduce the code duplication and constructed the Tactical Device Manager (TDM), upon which the concept of the URIM was built. The TDM partially used the XML Schemas that define ICD-0100 to create classes in an object-oriented programming language, in this case C#. Each sensor or camera could then be configured by filling in the various required fields of these classes, which could then be used in an easy and repeatable way to create ICD-0100 messages. The major advantage of this concept was that it only required a single module to generate ICD-0100 messages, which could then be used by each module that communicated directly with each sensor or camera. This reduced the creation time of a RIM by half and made the code much easier to debug and maintain.

The URIM took this concept a large step forward by making the data storage even more generic, implementing a publish/subscribe architecture and providing a multi-threaded interface that can process and handle data in a more flexible and efficient way. The URIM was designed to absolutely minimize code duplication (which exponentially adds risk) and be flexible enough to incorporate any known or future system quickly. It is currently the key interface to the JFPASS data fusion engine for the following items: 32 unattended ground sensors, 26 break-beam perimeter sensors, 4 chemical sensors, 3 day/thermal cameras, and 4 remotely operated weapon systems.

2. URIM APPLICATION OVERVIEW

The URIM (Figure 1), a Windows forms application developed using Microsoft Visual Studio 2008 and the programming language C#, maintains an in-memory database with a publish/subscribe architecture, which can be serialized and deserialized for long term storage. Each integrated system uses a protocol module to communicate with the underlying system (resource) and then store that data in a generic object in the database. Each protocol also requires a translator that converts the generic object of the database to the specific C2 protocol of the system (in the case of JFPASS this is ICD-0100). The URIM was designed to be as generic as possible to allow for future development as the needs of the system of systems change and adapt.

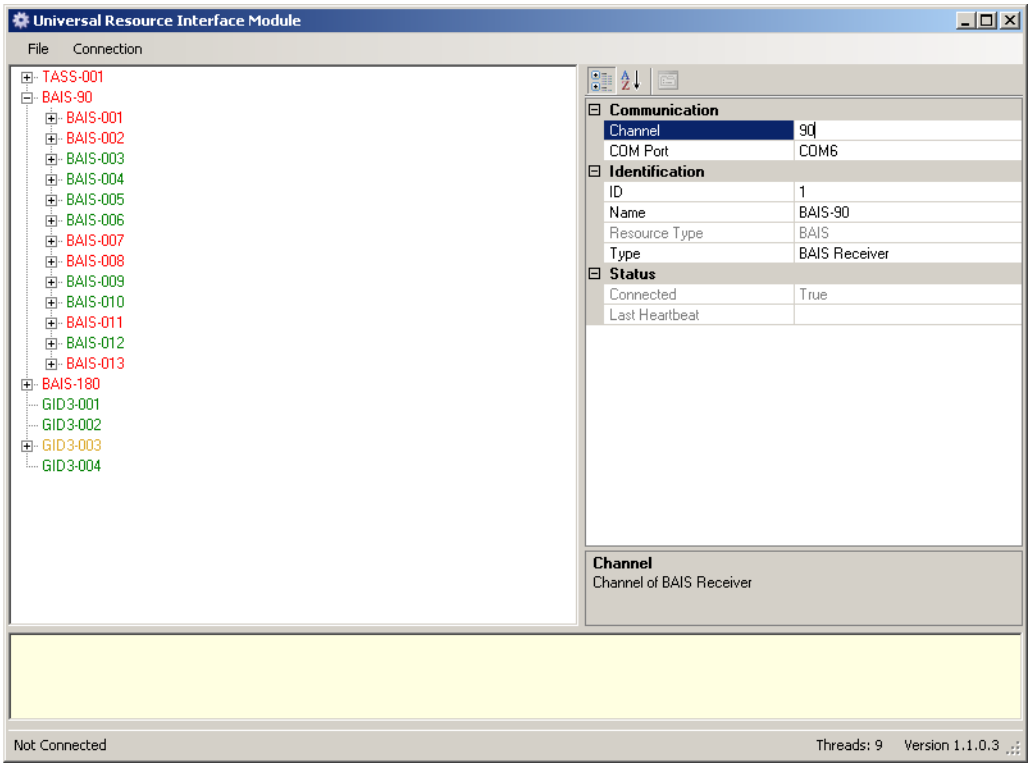

Figure 1: Screen shot of the URIM as used in the JFPASS. This interface is used by the technician for the initial configuration of the system. The left side contains the resource tree with the names of each resource and its children. The right side contains a property grid listing all available properties for a single resource. The bottom yellow area is for system messages (it is currently unused).

3. IN-MEMORY DATABASE

Force protection requires real-time data flow. In order to accomplish this, the URIM contains an in-memory database, which provides faster and more reliable performance than a disk database. Rather than complicating the software by linking it with a commercial or open-source database, a very simple construct was used to create an in-house, in-memory database.

The URIM contains a class called DataObject, which is an object with a Globally Unique Identifier (GUID), a list of properties (which can be any type of object), and a list of DataObjects (children). The intent was to make each data container as generic as possible so any protocol module would be able to store all data necessary to characterize any system. To fully characterize the status of a camera, for example, several key pieces of information are necessary: azimuth, field-of-view, tilt angle, latitude/longitude, altitude, just to name a few. However, a sensor does not necessarily have an azimuth or tilt angle, especially one that just sits in the ground and detects seismic vibrations. Furthermore, a weapon system may have all the information of a camera system, and also a few more things: firing mode, remaining ammunition, safety state, etc. Individual classes could be used to store each piece of data, such as a Camera, Weapon, etc., but what happens when one camera has features that another does not? A new type of camera class is required and so on until the database must support an unmanageable number of distinct classes. An object that contains nothing but a list of properties, however, requires no additional coding to handle any new device that may come along. In addition, a database that contains such DataObjects needs no upgrade to handle new devices.

A DataObject also contains a list of DataObjects that are children. This concept is partially rooted in the fact that the ICD-0100 uses platforms and nested platforms with device children to characterize a system. For example, a robot consists of the mobile platform itself, then it has certain devices, such as a camera or radar, but it may also have a full weapon system mounted, which could consist of a weapon, camera, range finder, etc. The ability to group individual devices in this family-style makes the system much easier to visualize in the abstract world of code and data, and thus easier to debug and manage. Because a DataObject can have children, and those children can have children, and so on, the organization of the data itself can be shaped to the needs of an individual system or interface. The database needs no new code to accommodate any new organizational needs.

The one requirement the URIM database imposes on its data is that it must be serializable, which is not a very strict restriction because any integer, double, string, character, or object consisting of such can be serialized. With all data available for serialization, the database can be saved at any time and written to a file, then loaded back in as needed.

4. PROTOCOL MODULES AND TRANSLATORS

4.1. Individual Systems

In addition to an in-memory database, the URIM contains protocol modules that perform direct communication with the various resources deployed at a given base or site. The URIM can currently communicate with the following systems: Battlefield Anti-Intrusion System (BAIS), Tactical Automated Security System (TASS), GID-3 24/7 Chemical Sensor, Networked Remotely Operated Weapon System (NROWS), and the Vaisala Weather Station.

4.1.1 BAIS

BAIS consists of unmanned ground sensors (UGSs) with seismic/acoustic detection capability, with optional IR and magnetic sensor modules that can be plugged in for further sensing ability. A BAIS sensor uses the information gathered through those various methods and performs analysis to classify the anomaly it has detected as a person, wheeled vehicle, tracked vehicle, or unknown. If it has IR or magnetic sensors, it can also sense direction of movement. The BAIS unit then transmits this "detection" via radio to a Hand-held Monitor (HHM). The HHM outputs the detection using a proprietary protocol via a serial port. This serial port is plugged into the network through a Serial-to-Ethernet converter. The URIM runs on a computer on the same network and parses incoming messages from the BAIS HHM.

These messages consist of 7 bytes: 2 bytes for the header, 2 bytes for the sensor ID, 1 byte for the detection type, and 2 bytes for the footer. The magnetic and IR modules have 2 IDs each, which are used to indicate direction. For example, BAIS-001 with both the IR and magnetic module would send messages with 5 different IDs. Assuming it uses IDs 1-5,

a message with ID 1 would indicate a seismic/acoustic detection. A message of ID 2 indicates a motion from left to right past the IR beam. A message of ID 3 indicates motion from right to left past the IR beam. The same concept is used for the magnetic sensor with IDs 4 and 5.

Once a message has been parsed and validated in the URIM, the data about the detection is stored in a DataObject (see Figure 2). It is also important to note that all information about each sensor and receiver is stored in DataObjects as well. The family organization is used, with the receiver being the parent, the sensors being children of the receiver, and detections being children of sensors (since a sensor can have many detections). In Figure 2 below, BAIS-90 is the receiver, BAIS-001 thru BAIS-013 are children, and you can see the 10 most recent detections. The use of these DataObjects in the URIM is discussed further in Section 5.

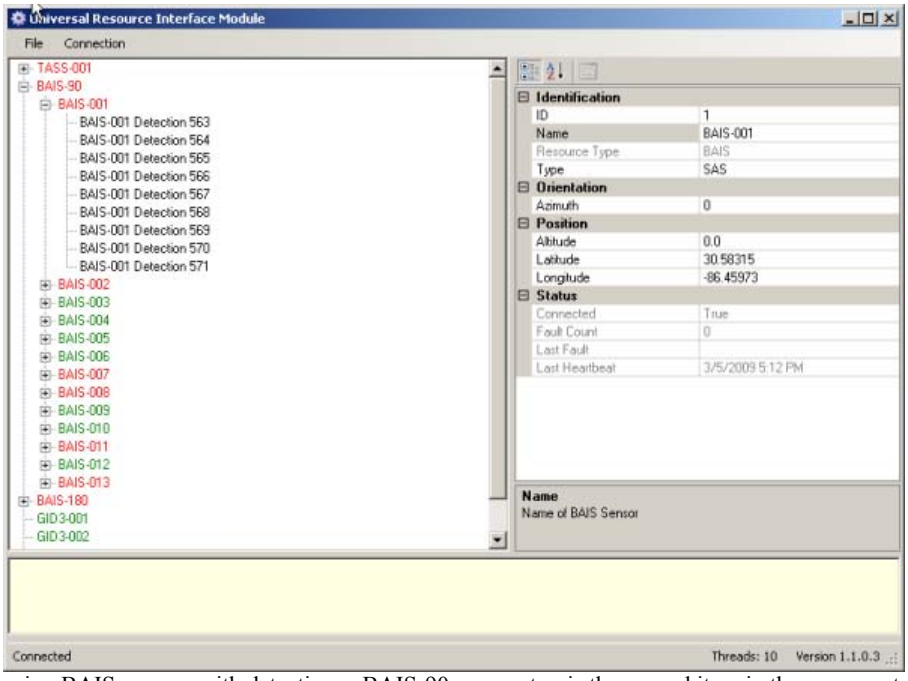

Figure 2: URIM showing BAIS sensors with detections. BAIS-90, a repeater, is the second item in the resource tree, with its children expanded. Its child, BAIS-001 (seismic/acoustic sensor), also has its children expanded, showing 9 separate detection instances. This means BAIS-001 has detected seismic/acoustic anomalies 9 times. Clicking each detection will bring up further details in the property grid, including time of detection and certainty. On the right, the properties of BAIS-001 are displayed, including its lat/lon and the date and time of the last heartbeat.

4.1.2 TASS

Already deployed around the world, the TASS system consists of various break-beam sensors and cameras. The sensors and cameras use different protocols, known to JFPASS as the TASS protocol and the ICD-001 protocol, respectively. The TASS sensors are set up on various strategic paths and perimeters around a base or site. When a beam is broken, the sensor sends a detection via radio to the TASS receiver. This box functions much like the HHM for the BAIS system and also has a serial output, which is converted to Ethernet.

The URIM also parses these messages, which are 15 bytes long and include a four-byte header, three-byte footer, various sensor identifying information, a checksum, and a single byte indicating the type of event. There can be tamper events (someone has broken into the sensor control box) and intrusion events (someone has broken the beam), among others. The detections generated by these messages are then stored in the URIM as DataObjects, just like the BAIS detections.

The TASS cameras use the ICD-001 protocol, which consists of messages with a header, footer, checksum, and then a plethora of other data which is used to request or pass various information about the camera. Unlike the sensors, the cameras have two-way communication so they can be controlled remotely, while at the same time reporting status such as azimuth and field-of-view. A camera and all its information are stored as a DataObject.

Figure 3: FLIR Sentry II Wide-scan Thermal Imager (WSTI). Commands and status information are sent to and from this camera using the ICD-001 protocol.

4.1.3 GID-3 24/7 Chemical Sensor

The GID-3 24/7 chemical sensor is a man-portable sensor placed inside a weatherproof stainless-steel box with a thermostat and a computer that provides an Ethernet interface. When polled, the GID-3 sends a status message containing 64 bytes, including a header, footer, checksum, and many pieces of sensor information, including the symbol for a chemical agent and the concentration detected.

4.1.4 NROWS

The NROWS system consists of a weapon, cameras, and a pan/tilt platform that can be mounted on a tripod, inside clamshell or popup housing, or on a mobile platform such as a robot. The system has a power box that supplies all power and a separate e-box that contains all the electronics for pan, tilt, firing, and other controls.

Figure 4: Networked Remotely Operated Weapon System. This particular setup has a single scope camera and a dummy shotgun mounted on the pan/tilt platform. The system is mounted on a mobile station.

The e-box has an Ethernet port, which makes it very easy to put on the network. The URIM connects to it and messages are passed back and forth using Extended Markup Language Remote Procedure Call (XML-RPC). The URIM stores all information for this platform as a single DataObject that contains nearly 100 properties to fully characterize the system. Figure 3 below shows about 25% of the properties of ROWS1, which is a trailer mounted pop-up weapon system.

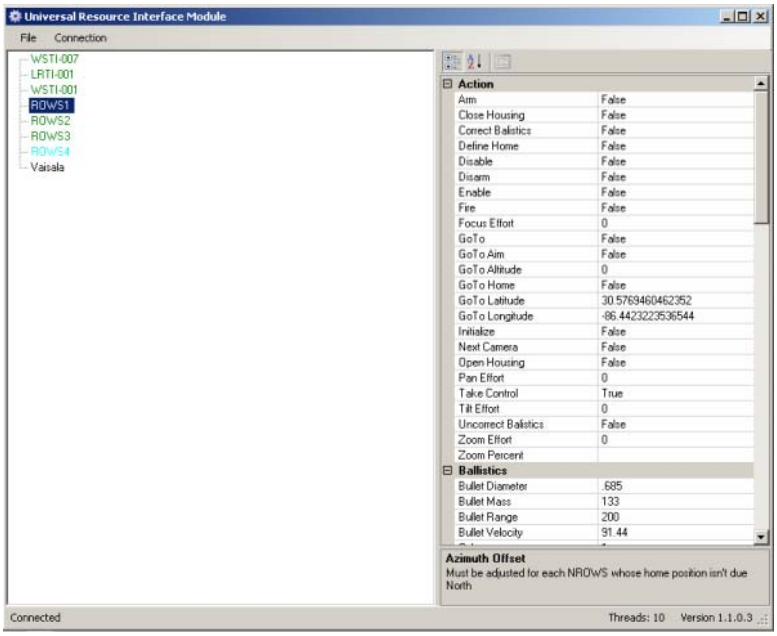

Figure 5: URIM showing just a few of the NROWS properties. Action properties are used to indicate whether an action should take place or give details of an action to be performed. Actions generally stay false because they are reset as soon as the action is complete. For example, Arm would be true briefly until the weapon reported that it has actually armed, then Arm would go back to false (while the Armed status property, not shown, would then be true).

4.1.5 Vaisala Weather Station

The Vaisala Weather Station has no moving parts but still provides accurate information about wind speed and direction, rainfall, temperature, and humidity. It outputs this data using a proprietary protocol through a serial port. The protocol is fairly simple and consists of strings with information in key/value pairs. The URIM reads in this weather information periodically and uses it to update the DataObject that stores all pertinent data. Other modules use this data, such as the weapon, which takes in wind information for ballistics correction (to make sure the bullets hit the target, even on windy days).

4.2 C2 Link

The C2 link is a module that communicates with the data-fusion engine. The main task of the URIM is to provide a single protocol interface to as many resources as possible. The URIM opens and maintains a socket connection with the data fusion engine and sends updates as they occur using the ICD-0100 format. The URIM also receives commands from the various C2 nodes and outputs them to the specific resources in a language they can understand. The C2 link module receives and parses messages in the ICD-0100 format, then calls translators where appropriate to perform the translation between the ICD-0100 format and the DataObjects that fully represent each system.

4.3 Translators

The translator concept is another feature of the URIM that allows for flexibility. A translator is an extension of a protocol module that provides the specific code to convert native DataObjects to ICD-0100. The JFPASS system does not use every possible feature of every resource. This is partially due to the fact that all features are not needed, but it is also partially a result of the limited time allotted to develop the system. By using translators, the URIM can provide for current needs while remaining open and flexible for future development.

That being said, translators provide the best possible conversion between DataObjects and ICD-0100 and separate the specific code from the generic code to keep the system flexible. As stated before, the C2 Link module calls these translators when they are needed. For example, a Wide-scan Thermal Imager (WSTI) has an azimuth, field-of-view, tilt angle, focus level, shutter state, auto-scan state, etc. However, the JFPASS system currently only uses the azimuth and field-of-view because the C2 and COP nodes currently cannot display any more information about the camera. Thus, the translator reads data from the DataObject and passes only the necessary information along in the ICD-0100 format. Also, when an ICD-0100 command message arrives, the C2 Link calls the translator to pass the command along to the protocol module.

4.4 Graphical User Interface (GUI)

The GUI of the URIM provides the basic functionality of displaying the various DataObject trees and allows the user to view and configure the individual resources. In some cases, the URIM GUI provides more control over a resource than the JFPASS system because it is more completely integrated. The NROWS platform, as shown in Figure 3 above, is a good example of this phenomenon.

In order to be as generic as possible, the URIM uses the PropertyGrid control, which simply displays all properties attached to an object, in this case, a DataObject. If a DataObject is selected in the tree, its properties are displayed in the PropertyGrid and they update once a second while the user is not editing any of the property fields. The PropertyGrid allows categorization of the properties, along with useful descriptions. It also makes validation possible so users cannot enter invalid data into a given field. The PropertyGrid also allows for read-only properties that display as gray and cannot be edited (this can be seen in Figure 2 in the properties under the Status category). This GUI provides simple and useful access for advanced configuration, as well as near real-time status.

5. PUBLISH/SUBSCRIBE ARCHITECTURE

Publish/subscribe methodology is a paradigm of communication. It is very much like object-oriented programming in the sense that it allows software developers to visualize and organize code in such a way that it makes sense to the human brain. An above-average person cannot understand a page full of ones and zeroes, but even a toddler can understand that a Dog can Bark(). In publish/subscribe, one entity publishes data and it is received by all other entities that are subscribed to that type of data.

The URIM is one particular implementation of Publish/Subscribe methodology. As described above, all data in the URIM is stored in generic DataObjects. In addition, the URIM has a class called a DataInterface. When a resource is added to the URIM, a new thread is called from the ThreadPool and the main function of a DataInterface begins to run. The important thing to note about a DataInterface is that it can subscribe to DataObject adds, deletes, and property updates.

A DataObject keeps a list of function pointers to those DataInterfaces that have subscribed to it. When a child is added or removed, or a property is changed, the DataObject "publishes" by calling each subscriber's function in a separate thread. For example, if there are 30 DataInterfaces subscribed to the NROWS DataObject, 30 threads are called from the ThreadPool to run 30 subscriber event handlers. This use of threading allows for asynchronous real-time action and parallel processing. In actual practice, there are usually only three subscribers to any given DataObject: the C2 Link, the GUI, and the DataInterface to which the DataObject belongs. There is also flexibility for other useful subscriptions. For example, the NROWS DataInterface subscribes to the Vaisala Weather Station DataObject and receives updated weather information whenever it is received and parsed by the URIM.

An example will now be supplied to provide a full visualization of the publish/subscribe action in the URIM. Figure 4 is provided initially to help visualize the example:

The GUI is initialized when the URIM application is run. A camera using the ICD-001 protocol is added to the URIM by right-clicking the open space and selecting the camera image. This calls a thread from the ThreadPool to run the DataInterface for the camera. The DataInterface calls into the ICD-001 protocol module for the startup tasks of setting up the DataObject with the appropriate properties and connecting it to the resource. The GUI subscribes to the DataObject upon creation and adds it as a top level object in the resource tree. The DataInterface running the ICD-001 code also subscribes to the DataObject. This decoupling allows the DataInterface to respond when other modules modify the DataObject such as the GUI or a translator.

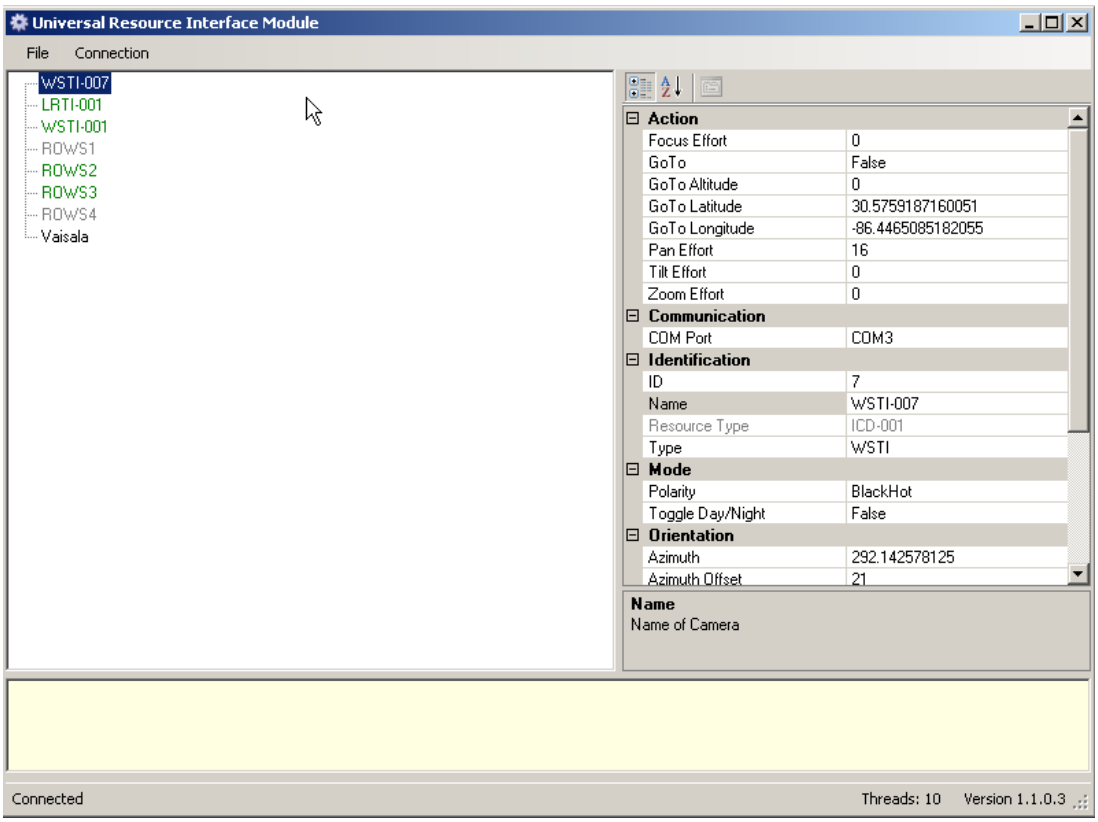

Figure 6: URIM GUI showing a WSTI camera and its properties

The technician then chooses the COM port in the PropertyGrid that corresponds with the location of the camera on the network. This can be seen in Figure 4 in the right column. The chosen COM port is COM 3. When a connection is made, the read-only Connected property (not visible in figure) changes from False to True in the GUI within a second of actual connection (the PropertyGrid reloads all properties at 1 Hz). The Azimuth and Field-of-View properties also change to reflect the true position of the camera (Azimuth is visible in Figure 4). The user then changes the name from the default to the more appropriate "WSTI-007." The GUI receives this publication and changes the name of the resource in the tree (note that the Name property in the PropertyGrid on the right corresponds with the name listed in the resource tree on the left.) Other resources are added in the same fashion and configured as appropriate.

The technician then clicks the Connection menu item and chooses Configure. This brings up a dialog with a PropertyGrid in which a user can enter an IP address and port. Upon closing, a DataInterface for the C2 Link is created and automatically subscribes to all existing DataObjects. In addition to connecting to the data fusion engine, it calls for an initial translation of each DataObject. It sends out the initial status report once connected and the data fusion engine is now aware of the all resources connected to the URIM and it passes the message along to all C2 and COP nodes.

An operator (likely from security forces), who now sees that WSTI-007 is available (among other resources), takes control of the camera and sends a pan and tilt effort (percentage of maximum available speed) by moving the joystick up and to the right. This message flows in ICD-0100 format from the C2 node to the data fusion engine, which passes it along to the URIM. The C2 Link module parses the data and then translates the ICD-0100 command into useful data in the WSTI-007 DataObject, in this case by changing the pan and tilt effort properties of the camera. The WSTI-007 DataObject then publishes the pan and tilt changes to the GUI, which displays them (in the figure, the pan effort property has a value of 16). It also publishes to the DataInterface maintaining the connection with the camera. The DataInterface then calls the protocol module, which constructs and sends a message in ICD-001 telling the camera to pan and tilt at the given effort rates.

The actual camera then pans and tilts, which causes a change in its azimuth. The DataObject publishes this change to all subscribers, including the GUI and C2 Link. The C2 Link then calls the translator to convert this status update to an ICD-0100 message and sends it to the data fusion engine, which disperses the information to all C2 nodes and COPs, each of which shows the new pose of the camera. This repeats until the joystick goes back to home position and pan and tilt values of 0 are sent out. The camera then stops moving, which causes updates to stop.

This method of publishing and subscribing data allows asynchronous updating for near real-time command and control of the various resources. It allows a single C2 Link module to receive and send all messages for the given protocol while the other interfaces need only work with generic DataObjects that have been set up specifically for the resources at hand. The translators provide further decoupling of generic and non-generic code and allow rapid integration.

6. FUTURE EFFORTS

The URIM has proven its utility in many hours of testing and experimental operation at Site C3 at Eglin Air Force Base in Florida. New resources have been rapidly integrated into the JFPASS system in a matter of a few days. The URIM is a stable and reliable product that will be highly utilized, in its current state, as part of the JFPASS system. The URIM is an excellent tool for quick integration into the JFPASS system and is robust enough to allow for full control of any asset. It is a light piece of software that can live on a small computer with Ethernet access or on a powerful machine, depending on the situational requirements. The URIM is an essential part of the JFPASS system.

However, there are ideas that could further improve the URIM and make it even more universal. Currently, to integrate a new resource into the URIM, a new .cs file (C# file) must be added to the project and the code must be rebuilt. However, it may be possible to fully characterize a given protocol using only configuration information. In this case, instead of having to write a full C# class and dropping it in, the integrator would be able to choose from common configuration options and supply advanced functionality where needed, all through a GUI which would then serialize the information for use in later operation. This concept of non-code integration would clearly require a large amount of design work. However, it would make the URIM much more universal and provide even faster integration for many new resources. It would also further maximize code reuse. As shown above, all systems that the URIM communicates

with have messages of a given length, with specific headers and footers. The function of receiving a message and parsing it could easily be specified by supplying the standard message length, the header and footer values, the connection method (TCP, RS-232, etc.), and a few other options.

REFERENCES

[1] D. Powell, G.A. Gilbreath, *Multi-robot Operator Control Unit (MOCU)*, SPIE Proc. 6230-67, Orlando, FL, 2006.

[2] T.A. Kramer, C.M. Barngrover, R.T. Laird, J.R. Cruickshanks, M. Dinh, *FIRRE Joint Battlespace Command and Control System for Manned and Unmanned Assets (JBC2S)*, SPIE Proc. 6230-82, Orlando, FL, 2006.